TAME: an Efficient Task Allocation Algorithm for Integrated Mobile Gaming

TAME: an Efficient Task Allocation Algorithm for Integrated Mobile Gaming

Tianzhu Zhang, *Member, IEEE,* Carla Fabiana Chiasserini, *Fellow, IEEE,* and Paolo Giaccone, *Senior Member, IEEE*

Abstract—We consider an integrated mobile gaming platform, in which the mobile device (e.g., smartphone) of a player can offload some game tasks toward a server as well as some neighboring mobile devices. The advantages of such a platform are manyfold: it can lead to an improved game experience, to a better use of energy resources, and, while offloading tasks to other mobile users, to the exploitation of the unused computing and storage resources of the mobile equipments, thus reducing the bandwidth and computing costs of the overall system. In this context, we formulate the problem of offloading the game computational tasks as an optimization problem that minimizes the maximum energy consumption across a set of mobile devices, under the constraints of a maximum response time and a limited availability of computation, communication and storage resources. In light of the problem complexity, we then propose a heuristic, called TAME, which is shown to closely approximate the optimal solution in all scenarios we considered. TAME also outperforms state-of-the-art algorithms under both synthetic and real scenarios, which have been devised based on a realistic and detailed energy consumption model for computation and communication resources. Our results, although tailored to mobile gaming, could be extended to other applications where it may be beneficial to offload computational and storage tasks through device-to-device communications, as enabled by Wi-Fi, Bluetooth, or the upcoming 5G technology.

Index Terms—Fog computing, online games, task offloading algorithms, optimization.

1 Introduction

Mobile Cloud Gaming (MCG) [1] offers the possibility of running sophisticated games on thin mobile devices by offloading heavy tasks to the cloud. In this way, mobile games can be accessed on any device from anywhere with a simple setup. During the last few years, MCG has sharply motivated the expansion of mobile game industry. According to the report by Newzoo [2], in 2016 game users generated $\$99.6$ billion of revenues, with an increase of 8.5% compared with 2015, and mobile games began to take a larger market share than their PC counterpart for the first time. Additionally, the Asia Pacific cloud gaming market is expected to witness a compound annual growth rate of 22% between 2016-2022 [3]. Finally, as a prevalent gaming model, MCG is widely supported by many famous online gaming platforms including Onlive [4], GaiKai [5] and GamingAnywhere [6].

In spite of the increasing popularity of MCG, several important challenges still need to be faced. First, offloading tasks to the remote cloud imposes extra communication latency, which may degrade users' quality of experience (QoE) and limit user coverage due to strict requirements on the response delay [7], [8]. Second, a large amount of bandwidth is required in order to guarantee high game quality, which in turn increases the costs for gaming service providers [1]. Third, cloud infrastructure requires more and more resources to cater the ever increasing demands of large scale mobile games (e.g., World of Warcraft) [9].

In this paper, we propose a general mobile gaming platform, named Integrated Mobile Gaming (IMG), which combines the available resources at both a game server and the neighbor mobile nodes, denoted as *mobile fog*, to run a game on behalf of the player's device. This new model shares MCG's idea of augmenting mobile devices with computation offloading, whereas it overcomes the intrinsic drawbacks of traffic offloading such as long response latency, wireless bandwidth consumption and limited available energy and computational resources. By partitioning a game into fine-granularity tasks and offloading part of them to either the game server or neighbor nodes cognitively, not only the player's local device can save energy but also the available resources of the network can be better utilized. Similar to [10], [11], we partition games into multiple tasks, at both object and method level, for the sake of offloading flexibility. Then we determine the offloading strategy by solving an integer linear programming (ILP) problem as well as by devising an approximation algorithm exhibiting excellent scalability.

As an example, consider the IMG scenario depicted in Fig. 1, including the player's device (denoted by n_0) and some mobile devices in proximity, a Point of Access (PoA) and a game server, located at the edge of the network.

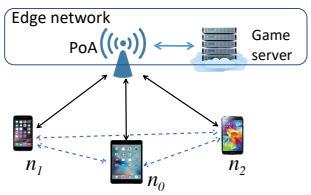

Fig. 1: Exemplifying scenario of Integrated Mobile Gaming (IMG).

• *All the authors are with the Electronics and Telecommunications Department, Politecnico di Torino, Italy. C.F. Chiasserini is also a Research Associate with IIET-CNR, Italy. E-mail: firstname.lastname@polito.it*

Notably, the PoA can be a standard Wi-Fi AP or can be the access node of a cellular network. Beside the connection with the game server through the PoA, each of the mobile devices can exploit device-to-device (D2D) communications with the neighboring nodes, enabled by Bluetooth or Wi-Fi Direct technologies. According to the IMG model, the game running on n_0 is firstly partitioned into tasks, part of which can be offloaded to the neighboring devices (e.g., n_1 and n_2) or the game server, without degrading gameplay experience. Through task offloading, we aim to minimize the maximum energy consumption across all mobile nodes, instead of just at the player's device, since, as shown in [12], the case of multiple players in the network minimizing unilaterally the energy cost for their own devices may end up with a lose-lose situation.

To the best of our knowledge, this is the first paper that exploits both edge computing and mobile fog computing for energy-efficient mobile gaming. In mode detail, we provide the following contributions: (i) we formulate the optimal energy-aware task offloading problem for mobile gaming under the IMG model; (ii) we devise an approximate algorithm, named Task Allocation with Minimal Energy cost (TAME), which is able to account for both computation and communication costs; (iii) we evaluate the performance of TAME under synthetic and realistic scenarios, and show numerically that it approximates very closely the optimal solution and outperforms other state-of-the-art offloading algorithms.

The remainder of the paper is organized as follows. Sec. 2 presents the system model of IMG, and describes the available computation and communication network resources as well as the game structure in terms of tasks and their interaction. Sec. 3 introduces the optimal energy-efficient task offloading problem, while Sec. 4 describes our approximate algorithm TAME. Sec. 5 presents the methodology used to investigate the system performance, which is then shown in Sec. 6. Finally, Sec. 7 discusses related work highlighting the novelty of our contribution.

2 System model for IMG

We first introduce the network graph $\mathcal{G}_N$, which describes the network topology and all its computation, communication and storage resources. Then we define the task graph $\mathcal{G}_T$, characterizing the game tasks and their dependency.

2.1 Network model

We consider the network topology depicted in Fig. 1. Each mobile device can communicate with a Point of Access (PoA), e.g., a Wi-Fi access point or a base station, and with its neighbor mobile nodes through any device-to-device (D2D) communication technology. We envision that the mobile game server is implemented as an NFV application in the mobile network operator infrastructure and connected to the PoA through a wired connection. Leveraging the information available at the PoA on the user presence, the server can provide the application at the player's device with the list of nodes that can be involved in the task offloading. Communication links between the player's device and its neighbors can be established using standard procedures for D2D communications [13]. Alternative solutions where the game server is located in the cloud can be envisioned as well.

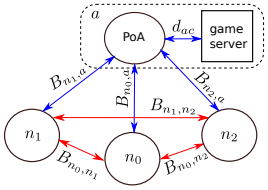

Fig. 2: A sample network graph $\mathcal{G}_N$ with three mobile nodes: one player's node n_0 and two neighbor nodes n_1 and n_2.

Let $\mathcal{N}$ be the set of all network nodes, comprising the mobile nodes and the PoA. We denote by n a generic network node; with an abuse of notation, we will use a, with $a \in \mathcal{N}$, to refer to both the PoA and the game server. We assume that the network evolves through a sequence of *temporal epochs*, each with duration of the order of tens of seconds or minutes. During each epoch, the communication capacity between pairs of nodes remains constant; indeed, we include in the $\mathcal{N}$ set only those neighbor nodes whose movement with respect to the player's device is negligible during an epoch.

Let $\mathcal{G}_N$ be the *network graph* representing our network during an epoch: each vertex corresponds to a network node while edges represent communication links between nodes in radio visibility. An example of $\mathcal{G}_N$ in the case of a network including three mobile users is depicted in Fig. 2, while Table 1 summarizes the notation we use to describe the network. Let n_0 be the game player's device. A generic mobile node n is characterized by computation capacity C_n, available storage S_n, and available energy E_n. The server node a has computation capacity C_a, while its available storage and energy are considered as unbounded. Let $B_{n,m}$ be the throughput between node n and m, and $B_{n,a}$ the throughput between node n and the PoA. We assume that the propagation delay from any mobile node to the PoA is negligible, and that the available bandwidth between the PoA and the game server is so high that the only contribution to the communication delay between the PoA and the server is due to the propagation delay, denoted with D_{ac}. Regarding the energy consumption, we define E_n^C as the per-clock-cycle energy cost due to computing at node n. With regard to the transmission from node n to node m, let $E_{n,m}^T$ and $E_{n,m}^R$ be the per-bit energy cost for the transmission and the reception of data, respectively, similarly to the models adopted in [12], [14].

2.2 Mobile game model

The software of a game can be partitioned into several tasks. Different levels of granularity for the definition of the tasks can be adopted. For example, in a shooter game, the collision detection could be a single task defined *at method level*, i.e., shared across the objects requiring the evaluation of collisions. Instead, the automata governing different entities interacting in the game (e.g., the enemies) could run as individual tasks *at object level*, i.e. one automata task for each entity. Finally, the artificial intelligence module, comprising a compound of multiple classes, could run as a single

TABLE 1: Network model notation

Symbol	Description
$\mathcal{G}_N$	Network graph
$\mathcal{N}$	Set of all network nodes
m, n	Generic node in the network, $m, n \in \mathcal{N}$
a	PoA and/or game server, $a \in \mathcal{N}$
n_0	Player's mobile device
$B_{n,m}$	Throughput from node n to m [bit/s]
$B_{n,a}$	Throughput between node n and the PoA [bit/s]
D_{ac}	Propagation delay between the PoA and the edge node [s]
$E^T_{n,m}$	Energy consumption for node n to transmit data to m [J/bit]
$E^R_{n,m}$	Energy consumption at node m to receive data from n [J/bit]
E^C_n	Processing energy consumption at node n [J/cycle]
C_n	Computation capacity at node n [Hz]
S_n	Storage availability at node n [bit]
E_n	Available energy at node n [J]

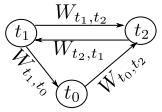

Fig. 3: A sample task graph $\mathcal{G}_T$, composed of one local task t_0 and two generic tasks t_1, t_2.

TABLE 2: Game model notation

Symbol	Description
$\mathcal{G}_T$	Task graph
$\mathcal{T}$	Set of all game tasks
$t,\ \tau$	Generic task of a game, $t, \tau \in \mathcal{T}$
t_0	Local task on n_0
F	Frame period duration [s]
s_t	Size of task t (code + state) [bit]
$\mathcal{O}$	Set of all the objects
s_o	Size of object o [bit]
c_t	CPU required by task t [cycles]
$W_{t,\tau}$	Data from task t to task τ [bit]

task. As another example, in a chess game with two self-playing opponents, the artificial intelligence module could run as a single task at class level (if shared between the two opponents), or as two distinct tasks at object level (i.e. one for each opponent). In Sec. 5.3.2, we will describe in details the tasks, defined at class level, that can be obtained for two real-world games.

Additionally, in real-time games, the software manages the calls of the tasks within a main event loop, which is responsible to update the entire state of the game, given the players' inputs, and to render the scene. The duration of the event loop is typically bounded by the scene *frame period* (e.g., 33 ms for 30 fps) with duration F.

For the sake of generality, we describe the tasks and their dependencies through a directed graph, called *task graph* and denoted by $\mathcal{G}_T$. Each vertex corresponds to a task, and an edge connects two vertices if there exists a dependency between the corresponding tasks, e.g., the output of the first task is used as input to the second one, or the second task runs only after the first has been completed. Fig. 3 shows a sample graph with 3 tasks; in Sec. 5.3.2, we will then discuss a methodology to obtain such task graph in the case of real-world games.

The notation we use for the task graph is summarized in Table 2. Let $\mathcal{T}$ be the set of all tasks, assumed to run in real-time. Let t_0 be a special task (possibly comprising a set of specific sub-tasks) that must run locally on the player's device n_0 (e.g., I/O processing, video rendering or decoding). Each task t is characterized by a computation requirement c_t, expressed as number of CPU cycles in a frame period, coherently with standard practice [12], [14]. Let $W_{t,\tau}$ be the average amount of data, measured in bits, sent from task t to task τ during a frame period. Some tasks may need some objects for their execution (e.g., the textures shown in the game, or the sounds), which must be stored at the nodes running the tasks. We assume that the game server has a copy of every object and that, at the beginning of each frame period, both the availability of objects at each node and the set of objects required by a task are known. With an abuse of notation, we say that $o \in n$ if object o is locally available at node n, and $o \in t$ if object o is required by task t. Finally, for simplicity we assume that only one portable device within the neighborhood is involved actively in the game playing activity.

TABLE 3: Decision and auxiliary variables in the optimization problem

Symbol	Description
$X_{t,n}$	1 if t is offloaded to n, 0 else
$\pi^{t,\tau}_{n,m}$	1 if task t on n sends data to task τ on m, 0 else
$\eta_{o,m,n}$	1 if n downloads object o from m, 0 else
$\nu_{n,o}$	1 if some task on n needs object o, 0 else

3 Problem Formulation

We formulate the problem of optimal energy-aware task offloading, under the system constraints presented in Sec. 2. Aim of the problem is to allocate each task to the most suitable node, so that the maximum energy consumption across the mobile nodes, including the player's one, is minimized. This ensures fairness in the energy toll requested to the mobile nodes for the game execution, and it can also be easily adapted to the case where the player's device is supposed to pay a higher energy toll than its neighbor devices. Notably, a different optimization objective could lead to a lower total energy consumption, but with very unfair allocation of the tasks across the nodes, which may be unacceptable for the user devices.

Input to the optimization problem are the network model, described by $\mathcal{G}_N$, and the game model, described by $\mathcal{G}_T$. The main decision variable is the binary variable $X_{t,n}$, which takes 1 iff task t is offloaded to node n and zero otherwise, with $X_{t,a} = 1$ denoting that the task is allocated to the game server. The related decision and auxiliary variables are summarized in Table 3. The objective function is:

$$\min \max_{n \in \mathcal{N} \setminus \{a\}} \epsilon_n \tag{1}$$

where ϵ_n is the energy cost on node n (excluding the PoA) due to the game execution, during a given frame period. As detailed in Sec. 3.1, ϵ_n accounts for the energy consumption due to task processing as well as to the communication required for task offloading, retrieval of non-local objects and data exchange between tasks. In addition, our optimization framework accounts for the performance perceived by the user, defined as response delay of the game software. Indeed, a game with high response delay or low frame rate cannot guarantee accurate and fluent game scenes, thus in our optimization problem, we constraint the response time to be below a given threshold. This performance metric is defined in detail in Sec. 3.2, while the system constraints

under which the optimization problem should be solved are reported in Sec. 3.3.

3.1 Mobile node energy consumption

The total energy cost for any mobile node n, with $n \in \mathcal{N} \setminus \{a\}$, in the given frame period is the summation:

$$\epsilon_n = \epsilon_n^P + \epsilon_n^C + \epsilon_n^M + \epsilon_n^O \tag{2}$$

Each term is defined below.

Task processing energy consumption ϵ_n^P. The total energy consumption of mobile node n, due to the computation of the tasks that node n hosts, is given by:

$$\epsilon_n^P = \sum_{t \in \mathcal{T}} X_{t,n} \cdot E_n^C \cdot c_t \tag{3}$$

where E_n^C denotes the per-clock-cycle energy cost of node n and c_t is the required number of CPU cycles for task t. Thus, $X_{t,n} \cdot E_n^C \cdot c_t$ is the energy consumption due to t being offloaded to n.

Task communication energy consumption ϵ_n^C. Each mobile node transmits and receives data from other nodes on behalf of the hosted tasks. The corresponding energy consumption at the generic node n is evaluated as follows:

$$\epsilon_n^C = \sum_{\substack{t,\tau \in \mathcal{T} \\ t \neq \tau}} \sum_{m \in \mathcal{N} \setminus \{n\}} W_{t,\tau} \cdot (\pi_{n,m}^{t,\tau} \cdot E_{n,m}^T + \pi_{m,n}^{t,\tau} \cdot E_{m,n}^R) \tag{4}$$

where we recall that $W_{t,\tau}$ represents the amount of data that task t sends to task τ. The first term within the summation denotes the energy used by node n to transmit to m (i.e., $E_{n,m}^T$), whenever t is running on n and τ on m (i.e., $\pi_{n,m}^{t,\tau} = 1$). Similarly, the second term is the energy used by n, running τ, to receive data from task t running on m (i.e., $E_{m,n}^R$). Notably, we consider also the reception of data from the PoA (i.e., $m = a$).

Task migration energy consumption ϵ_n^M. In the case of task migration from the player's node n_0 to another node m (possibly including the game server a), the energy cost at n_0 due to the task transmission is:

$$\epsilon_{n_0}^M = \sum_{t \in \mathcal{T}} \sum_{m \in N \setminus \{n_0\}} s_t \cdot X_{t,m} \cdot E_{n_0,m}^T \tag{5}$$

where s_t denotes the size of task t. Similarly, the energy cost at a generic destination node n receiving a task from n_0 is:

$$\epsilon_n^M = \sum_{t \in \mathcal{T}} s_t \cdot X_{t,n} \cdot E_{n_0,n}^R \qquad n \neq n_0, a \tag{6}$$

Note that we excluded the energy consumption at the PoA (i.e., $n \neq a$), since the PoA operates without energy limitations.

Object retrieval energy consumption ϵ_n^O. Since tasks may need to load some objects as input for execution, the nodes hosting the tasks are in charge of downloading the required objects from other nodes if not locally available. As a result, a node may need to transfer its locally available objects to others, or receive objects from other nodes. Thus, any $n \in \mathcal{N} \setminus \{a\}$ may experience the following energy consumption:

$$\epsilon_n^O = \sum_{o \in \mathcal{O}} \sum_{m \in \mathcal{N} \setminus \{n\}} s_o \cdot (\eta_{o,m,n} \cdot E_{m,n}^R + \eta_{o,n,m} \cdot E_{n,m}^T) \tag{7}$$

where s_o denotes the size of object o and $\eta_{o,m,n}$ indicates whether n needs to download object o from m or not. In particular, the two terms in the above equation represent the total energy cost for node n to, respectively, retrieve objects from others and transmit objects to others.

3.2 Response delay

The response delay δ experienced by the player in a given frame period is given by:

$$\delta = \delta^M + \delta^R + \delta^P + \delta^E \tag{8}$$

where each term is described below.

Task migration delay δ^M. The tasks that are offloaded from n_0 to other nodes or to the game server for remote execution, require to be transmitted; thus, they experience some migration latency given by:

$$\delta^M = \sum_{t \in \mathcal{T}} \sum_{n \in \mathcal{N} \setminus \{n_0, a\}} \left(\frac{X_{t,n} \cdot s_t}{B_{n_0,n}} + X_{t,a} \cdot \left(\frac{s_t}{B_{n_0,a}} + D_{ac} \right) \right) \tag{9}$$

The first term in (9) is the transmission time of task t, of size s_t, from n_0 to another mobile node n, given that the expected throughput between the two nodes is $B_{n_0,n}$. The second term in (9) is instead the migration delay from n_0 to the game server, given the transmission time to the PoA (i.e., $s_t/B_{n_o,a}$) and the propagation delay D_{ac} from the PoA to the game server.

Task processing delay δ^P. The processing time of task t, requiring c_t cycles and running on node n with C_n processing capability, is c_t/C_n. Notably, in the case the task runs in the server, this turns out to be c_t/C_a. The overall execution time depends on the degree of parallelism allowed to run the tasks. If we assume that all tasks are executed sequentially, the total task processing delay δ_{Seq} is:

$$\delta_{\text{Seq}} = \sum_{t \in \mathcal{T}} \sum_{n \in \mathcal{N}} \frac{X_{t,n} \cdot c_t}{C_n}$$

Instead, if all tasks are executed in parallel, the total task delay δ_{Par} is the maximum among all the processing nodes:

$$\delta_{\text{Par}} = \max_{n \in \mathcal{N}} \sum_{t \in \mathcal{T}} \frac{X_{t,n} \cdot c_t}{C_n}$$

We expect that in realistic scenarios some tasks can be executed in parallel and others sequentially. As a result, the actual task processing time is bounded as follows:

$$\delta_{\text{Par}} \leq \delta^P \leq \delta_{\text{Seq}} \tag{10}$$

In the following, for a worst case design, we will consider $\delta^P = \delta_{\text{Seq}}$.

Task communication delay δ^E. As shown in Fig. 3, tasks may need to exchange data, introducing communication latency. For a worst case design, we assume sequential communications, and thus the task communication delay can be formulated as follows:

$$\delta^E = \sum_{\substack{t,\tau \in \mathcal{T} \\ t \neq \tau}} \left(\sum_{\substack{n,m \in \mathcal{N} \\ n \neq m,a}} \frac{W_{t,\tau} \cdot \pi_{n,m}^{t,\tau}}{B_{n,m}} + \sum_{n \in \mathcal{N} \setminus \{a\}} \left(\pi_{a,n}^{t,\tau} + \pi_{n,a}^{t,\tau} \right) \cdot \left(\frac{W_{t,\tau}}{B_{n,a}} + D_{ac} \right) \right) \tag{11}$$

The first term in (11) includes the duration of the transmission from task t to task τ occurring from mobile node n to mobile node m, and equal to $W_{t,\tau}/B_{n,m}$, whenever such transmission occurs (i.e., $\pi^{t,\tau}_{n,m} = 1$). The second term in (11) considers the communications between the game server and a mobile node. It includes the transmission time $W_{t,\tau}/B_{n,a}$ from/to the PoA and the propagation delay D_{ac} between PoA and the game server whenever such communication occurs, either from the mobile node to the game server (i.e., $\pi^{t,\tau}_{n,a} = 1$) or vice versa (i.e., $\pi^{t,\tau}_{a,n} = 1$).

Object retrieval delay δ^R. We assume that objects, if needed, are retrieved sequentially, thus the total object retrieval delay is formulated as follows:

$$\delta_R = \sum_{o\in\mathcal{O}} \sum_{n\in\mathcal{N}\setminus\{a\}} \sum_{m\in\mathcal{N}\setminus\{n,a\}} \left(\frac{\eta_{o,m,n}\cdot s_o}{B_{m,n}} + \eta_{o,a,n}\cdot\left(\frac{s_o}{B_{n,a}} + D_{ac}\right)\right) \quad (12)$$

The first term in (12) represents the total delay $s_o/B_{m,n}$ for mobile node n to retrieve object o from mobile node m whenever convenient (i.e., $\eta_{o,m,n} = 1$). The second term refers to the download time of o from the game server, i.e., between the game server and the PoA (i.e., D_{ac}) and between the PoA and node n (i.e., $s_o/B_{n,a}$), whenever this case happens (i.e., $\eta_{o,a,n} = 1$). We recall that the game server has a copy of any object, thus objects are never uploaded to the server.

3.3 Constraints

Given the variables defined above, the IMG system is subject to the constraints listed below.

Maximum response delay. The execution of all tasks related to a given frame period needs to be completed within the frame itself:

$$\delta \le F \quad (13)$$

Task mapping. Each task can only be offloaded to either a mobile node or the game server, i.e.,

$$\sum_{n\in\mathcal{N}} X_{t,n} = 1 \qquad \forall t \in \mathcal{T} \quad (14)$$

Additionally, since t_0 is the local task that must run on node n_0, we force $X_{t_0,n_0} = 1$.

Task communication. The auxiliary binary variable $\pi^{t,\tau}_{n,m}$ is related to the main decision variable $X_{t,n}$ as follows:

$$\pi^{t,\tau}_{n,m} = \begin{cases} X_{t,n}\cdot X_{\tau,m} & \text{if } W_{t,\tau} > 0 \\ 0 & \text{else} \end{cases} \quad (15)$$

i.e., a data transfer between tasks t and τ running, respectively, on nodes n and m, occurs only if τ gets as input $t's$ output, and t and τ are actually assigned to the two mobile nodes. In order to obtain linear constraints, we can equivalently express (15), when $W_{t,\tau} > 0$, as:

$$\pi^{t,\tau}_{n,m} \ge (X_{t,n} + X_{\tau,m} - 1), \quad \pi^{t,\tau}_{n,m} \le X_{t,n}, \quad \pi^{t,\tau}_{n,m} \le X_{\tau,m}$$

Object demand. Let $\nu_{n,o}$ be a binary variable such that $\nu_{n,o} = 1$ iff one or more tasks on node n need object o to be executed. Thus, the value of $\nu_{n,o}$ should be such that:

$$\sum_{t|o\in t} X_{t,n} \le K\cdot\nu_{n,o} \quad (16)$$

$$\sum_{t|o\in t} X_{t,n} \ge \nu_{n,o} \quad (17)$$

where K is a large enough constant. Indeed, if at least one task t running on node n needs object o (thus, $X_{t,n} = 1$ and $o \in t$) then the summation across all tasks in (16) implies $\nu_{n,o} = 1$; otherwise, (17) imposes $\nu_{n,o} = 0$. We remark that (17) is superfluous, since the energy minimization will prevent the case $\nu_{n,o} = 1$ to occur whenever task t does not require object o.

Object retrieval. If node n hosts tasks requiring an object that is not locally available, then one copy of the object must be downloaded from other nodes, i.e., for any $o \notin n$:

$$K\cdot\sum_{m\in\mathcal{N}\setminus\{n\}} \eta_{o,m,n} \ge \nu_{n,o} \quad (18)$$

An object is downloaded by n from other nodes (i.e., $\sum_{m\in\mathcal{N}\setminus\{n\}}\eta_{o,m,n} = 1$) iff it is not locally available (i.e., $o \notin n$) and it is needed by the tasks running on node n (i.e., $\nu_{n,o} = 1$), as modeled in (18). Similarly to (17), we remark that $\nu_{n,o}$ will take 0 zero unless (18) is satisfied.

3.3.1 Resource capacity

Since mobile devices have limited battery and computation capabilities, they must have enough resources in order to host tasks. For each mobile device, we define four kinds of resource capacity constraints: CPU, bandwidth, storage and energy constraints, as described below.

Node CPU constraint. Each node n (either a mobile node or the game server) must have enough CPU to satisfy the computation requirements of all the tasks it hosts in the given frame period. Thus, for any $n \in \mathcal{N}$:

$$\sum_{t\in\mathcal{T}} X_{t,n}\cdot c_t \le F\cdot C_n \quad (19)$$

The left side is the summation of CPU cycles required by all tasks running on node n, while the right side is the total number of CPU cycles available at the node.

Node bandwidth constraint. In the given frame period, each node n (either a mobile node or the PoA/server) must have enough bandwidth to support the data exchange between the local tasks and the remote tasks running on other nodes. Hence, for any $n, m \in \mathcal{N}$:

$$\sum_{\substack{t,\tau\in\mathcal{T}\\ t\neq\tau}} W_{t,\tau}\cdot(\pi^{t,\tau}_{n,m} + \pi^{t,\tau}_{m,n}) \le F\cdot B_{n,m} \quad (20)$$

The left side of (20) is the total amount of data that node n transmits/receives to/from m (i.e., $\pi^{t,\tau}_{n,m}$ and/or $\pi^{t,\tau}_{m,n}$ are equal to 1). The right side of (20) is the total amount of data node n can exchange with m, in the given frame period.

Node storage constraint. In a given frame period, each mobile node n has to store the objects needed by its local

tasks, i.e., for any $n \in \mathcal{N} \setminus \{a\}$, the total size of stored objects cannot be larger than the available storage at n:

$$\sum_{o \in \mathcal{O}} \nu_{n,o} \cdot s_o \leq S_n \tag{21}$$

Note that we implicitly assume non-persistent storage, and we expect that non-needed objects can be deleted by node n at the beginning of a frame period whenever $\nu_{n,o} = 0$ and the storage is full. We stress however that the set of objects stored by a node at the beginning of one frame period is given, and used as input to the optimization problem for the current frame period.

Node energy constraint. The energy consumption ϵ_n cannot be larger than the available energy E_n of a mobile node n, during the frame period, i.e., for any $n \in \mathcal{N} \setminus \{a\}$:

$$\epsilon_n \leq E_n \tag{22}$$

Recall that the game server has CPU and bandwidth limitations only.

3.4 Problem complexity

The above ILP formulation involves four kinds of decision variables and six kinds of constraints. It can be shown that the total number of variables grows as $O(|N|^2|\mathcal{O}|, |T|^2|N|^2)$, and the total number of constraints grows as $O(|T||N|, |\mathcal{O}||N|)$. According to [15], for a linear programming problem with v variables and α constraints, the complexity of an ILP solver is $O(2^{2^v}\alpha)$. Thus, the final complexity to solve our optimization problem is:

$$\begin{cases} O(2^{2^{|\mathcal{T}|^2 \cdot |\mathcal{N}|^2}} |\mathcal{T}||\mathcal{N}|) & \text{for } |\mathcal{T}| > |\mathcal{O}| \\ O(2^{2^{|\mathcal{O}| \cdot |\mathcal{N}|^2}} |\mathcal{O}||\mathcal{N}|) & \text{for } |\mathcal{T}| \leq |\mathcal{O}| \end{cases}$$

which underlines that the problem complexity greatly increases with the number of tasks and mobile nodes.

4 The TAME algorithm

The optimization problem presented in Sec. 3 can be solved with ILP solvers in the case of small problem instances, but, due to its high complexity, it cannot scale to large network/game instances. We therefore devise a low-complexity, greedy algorithm, named Task Allocation with Minimum Energy (TAME), which is amenable to an implementation in mobile user devices and can provide a swift solution to the problem of computational task offloading. TAME is inspired by the well-known greedy BestFit approximation algorithm, which has low complexity and has been shown to be very efficient in the allocation of virtual machines in cloud computing systems [16].

At each iteration, TAME assigns a task to the node that minimizes the energy cost. This cost depends on both communication and computation. To minimize the complexity when evaluating the energy costs, TAME identifies the dominant factor (either computation or communication) contributing to the total energy cost, based on all unallocated tasks. The task to offload is chosen according to the dominant factor, while ensuring compatibility with the resources (computation, communication and storage) that are available locally and at the neighbor nodes. In other words, at each iteration TAME adapts its "energy-awareness" to the most relevant energy contribution. It is worth mentioning that knowledge about the neighbors' resources can be acquired by the player's node through the assistance of the game server.

Algorithm 1 TAME algorithm

Require: $\mathcal{G}_T, \mathcal{G}_N$
1: $X_{t,n} = 0, \forall t \in \mathcal{T}, \forall n \in \mathcal{N}$ ▷ Init with no allocated task
2: $X_{t_0,n_0} = 1$ ▷ Allocate local task/s to n_0
3: $\Omega = \mathcal{T} \setminus \{t_0\}$ ▷ Init set of unallocated tasks
4: **while** $\Omega \neq \emptyset$ **do** ▷ For all unallocated tasks
5: $\epsilon_{\text{cpu}} = \hat{E}^C \cdot (\sum_{t \in \Omega} c_t)$ ▷ Energy due to computation
6: $\epsilon_{\text{com}} = (\hat{E}^T + \hat{E}^R) \cdot (\sum_{t,\tau \in \Omega, \tau \neq t} W_{t,\tau})$ ▷ Energy due to comm.
7: **if** $\epsilon_{\text{cpu}} \geq \epsilon_{\text{com}}$ **then** ▷ Computation energy is dominant
8: $t^* \leftarrow \arg\max_{t \in \Omega}\{c_t\}$ ▷ Task with highest computation requirement
9: $\tau^* = \{\}$ ▷ Task τ^* is undefined
10: **else** ▷ Communication energy is dominant
11: $t^*, \tau^* \leftarrow \arg\max_{t,\tau \in \Omega}\{W_{t,\tau} + W_{\tau,t}\}$ ▷ Task pair with highest communication requirement
12: **for all** $n \in \mathcal{N}$ **do** ▷ Find most energy-convenient node
13: assume t^* (and, possibly, τ^*) is offloaded to n
14: **if** all the constraints are satisfied **then**
15: $\hat{e}_n \leftarrow$ maximum energy cost across the mobile nodes if t^* (and, possibly, τ^*) is offloaded to n
16: **else**
17: $\hat{e}_n \leftarrow \infty$ ▷ n is not suitable
18: $n^* = \arg\min_{n \in \mathcal{N}}\{\hat{e}_n\}$ ▷ Find the node with minimum energy
19: **if** $\epsilon_{n^*} < \infty$ **then** ▷ A feasible choice exists
20: $X_{t^*,n^*} = 1$ ▷ Offload t^* on n^*
21: $\Omega = \Omega \setminus \{t^*\}$ ▷ t^* will not be further considered
22: **if** $\tau^* \neq \{\}$ **then** ▷ If τ^* is defined
23: $X_{\tau^*,n^*} = 1$ ▷ Offload τ^* on n^*
24: $\Omega = \Omega \setminus \{\tau^*\}$ ▷ t^* will not be further considered
25: **else**
26: **return** ▷ Task offloading is not possible
27: **return** X

The pseudocode of TAME is presented in Algorithm 1, which takes both the task graph $\mathcal{G}_T$ and the network graph $\mathcal{G}_N$ as input and returns the task allocation $\mathcal{X} = \{X_{t,n}\}$. After initialization, TAME allocates the local tasks t_0 to the player's device n_0 (line 2). Then it considers the remaining tasks iteratively, until all of them have been allocated (lines 4-26). At each iteration, TAME estimates the total energy cost due to computation and communication, for all the unallocated tasks. This allows identifying the major contribution to the energy cost, and, based on that, the optimal task allocation. In more detail, let $\hat{E}^C$ be the energy consumption corresponding to one computation unit, averaged across all the mobile nodes, then ϵ_{cpu} is the estimated total energy cost due to computation for all unallocated tasks (line 5). Likewise, let $\hat{E}^T$ and $\hat{E}^R$ be the energy consumption due to the transmission and reception, respectively, averaged across all possible pairs of mobile nodes. Then ϵ_{com} is the estimated total energy cost due to communication between all unallocated tasks (line 6). If the dominant energy contribution is due to computation, the unallocated task with the maximum energy (t^*) is selected (line 8). Otherwise, the dominant energy is due to communication, thus the pair of unallocated tasks with the maximum communication cost (t^* and τ^*) are selected. After having selected the task (or the pair of tasks) to allocate, all the

mobile nodes are considered as candidates to host such task/s (line 12-17). When a generic candidate node n is considered, the constraints on the maximum response delay and on the CPU, bandwidth, storage and energy resources are evaluated (as detailed in Sec. 3.3) If all constraints are satisfied (line 14), then $\hat{e}_n$ records the maximum energy cost among all the nodes, assuming that t^* (and possibly τ^*) is allocated on the node under consideration. Among all nodes for which the allocation is feasible, TAME selects the node n^* for which the cost $\hat{e}_{n^*}$ is minimum (line 18). Finally, t^*, and possibly τ^*, are offloaded to node n^* (lines 20-24).

Note that, since TAME greedily allocates tasks in each iteration, at some point it may happen that the response delay constraint cannot be met under the current partial task allocation thus resulting into an allocation failure. To solve this issue, TAME adopts a worst case prediction approach: when it verifies the response delay constraint (line 14), it assumes that for the future iterations the unallocated tasks will run in the player's node n_0. All nodes that fail to satisfy this response delay constraint are excluded from further inspection. So doing, TAME minimizes the probability of task offloading failure.

5 Performance evaluation methodology

We now introduce the methodology adopted to assess the performance of our scheme and to compare it against state-of-the-art solutions. In particular, Sec. 5.1 describes the two algorithms we use as benchmarks for the TAME algorithm, while Sec. 5.2 and Sec. 5.3 present, respectively, the network scenarios and the network task graphs used in our experiments.

5.1 Benchmark schemes

We evaluate the performance of our TAME algorithm against the following three approaches: (1) BestFit considers the tasks in decreasing order of CPU requirements and offloads each task to the node with the minimum available CPU resource, thus consolidating the tasks into the minimum number of mobile nodes such that the system constraints are met. As an example, assume that all mobile nodes have the same CPU capacity: since the local task t_0 is initially allocated to node n_0, all other tasks will be allocated into n_0 until possible. Then BestFit will assign tasks to the node with maximum available CPU till its capacity is saturated, and it will proceed in this way till no further allocation is possible. It follows that often BestFit does not offload any task. (2) GraphMerge [17] is based on the idea of combining tasks with low computation requirements and high communication cost into a "super-task", which is then allocated to the most suitable node. This is equivalent to merging nodes in the task graph, and it has the advantage of nullifying the energy cost due to communication between tasks that fall within the same super-task since they will be co-located in the same physical node. (3) Optimal solves optimally the ILP optimization problem formulated in Sec. 3.

TAME, BestFit and GraphMerge are implemented in Python, while in Optimal the solution is obtained by using the Gurobi solver [18]. We evaluate the *approximation ratio* of each algorithm defined as the ratio of the cost function $\max_{n\in\mathcal{N}\setminus\{a\}} \epsilon_n$ (as defined in (1)) obtained through the algorithm, to that of Optimal. Clearly, by construction, the approximation ratio is always equal or greater than one.

TABLE 4: Experimental setting for the network model $\mathcal{G}_N$

Parameter	Value
$\lvert\mathcal{N}\rvert$	$\{4, 6, 8, 10, 12\}$
$B_{n,a}$	$\{10, 20, 100\}$ Mbps
$B_{n,m}$	2 Mbps (Bluetooth), 20 Mbps (Wi-Fi Direct)
S_n	$\{16, 32, 64, 128\}$ GB
C_n	0.8 - 2.7 GHz, for $n \neq a$
C_a	3 GHz
E_n	18000 J $= 1350$ mAh
s_o	1000 bytes
E_n^C	Randomly chosen from Table 5
$E_{n,m}^T$, $E_{n,m}^R$	Randomly chosen from Table 6
D_{ac}	$\{1, 20, 50\}$ ms
$\lvert\mathcal{O}\rvert$	$\{0, 10\}$ objects

TABLE 5: Energy cost due to computation

Processor name	Max power [W]	Max frequency [GHz]	E_n^C [nJ/cycle]
Exynos 5433	0.22	1.5	0.147
Exynos 7420	1.29	2.1	0.614

5.2 Network scenarios

To generate the network model described by $\mathcal{G}_N$, we consider a network scenario characterized by the parameters listed in Table 4. The chosen CPU clock frequencies are derived from commercial products: C_a is coherent with Intel i7/i9 CPU cores, and C_n with Samsung Exynos processors (considered later in Table 5) as well as with state-of-art Qualcomm Snapdragon processors. The values we selected as size of storage resources are compatible with the maximum size available in high-level smartphones (e.g., iPhone 8). We vary the total number of network nodes $|\mathcal{N}|$ (which we recall it includes the player's node n_0 and the game server a) from 4 to 12; this corresponds to a number of neighbor mobile nodes for n_0 varying from 2 to 10. Node n_0 is connected to the neighbor nodes through a Wi-Fi Direct or Bluetooth interface, while it communicates with the PoA through a Wi-Fi interface. The delay D_{ac} varies from 1 ms (which is very small and typical of edge computing systems) to 50 ms (which is reasonable for a game server in the cloud [19]). The energy consumption for computation (E_n^C) and communication ($E_{n,m}^T$, $E_{n,m}^R$) are all derived from real-world mobile processors and wireless modules, as detailed below.

Computation energy cost E_n^C. We estimate the energy consumption per clock cycle by dividing the nominal maximum power of a mobile processor by its maximum CPU frequency. Specifically, we consider two mobile processors: the Exynos 5433 equipping the Samsung Note 4 [20], and the Exynos 7420 equipping the Samsung Galaxy S6 [21]. The resulting values are shown in Table 5.

Communication energy costs $E_{n,m}^T$ and $E_{n,m}^R$. We estimate the energy cost to transmit ($E_{n,m}^T$) and receive ($E_{n,m}^R$) data by computing the per-bit energy consumption through the formula $V \cdot I/r$ applied to the considered communication chipset, where V is the voltage supply and I is the current when transmitting/receiving at the physical data rate r. We consider different Wi-Fi and Bluetooth chipsets, and we set

TABLE 6: Energy cost due to communication.

Technology	Module name	V [V]	$I_T; I_R$ [mA]	$E^T; E^R$ [nJ/bit]
WiFi-Direct	SPWF01SA	3.3	243.0; 105.0	14.8; 6.4
	QFM-2202	3.3	201.2; 66.7	12.3; 4.1
	QCA6234	3.3	250.0; 69.0	15.3; 4.2
	WGM110	4.8	246.0; 81.0	22.0; 7.2
Bluetooth	RN-42(N)	3.3	30.0; 30.0	99.0; 99.0
	HC-06	3.3	8.0; 8.0	26.4; 26.4
	BT4 BLE	3.3	8.5; 8.5	28.0; 28.0
	BT24	3.3	29.0; 29.0	95.0; 95.0
	SPBT2632	2.5	23.0; 23.0	57.5; 57.5

TABLE 7: Experimental setting for synthetic task graphs

Parameter	Value
$\lvert\mathcal{T}\rvert$	$\{4, 6, 8, 10\}$ tasks
$d_{\mathcal{T}}$	2
c_t	Uniformly distributed in $[10^5, 10^6]$ clock cycles
$W_{t,\tau}$	Exponentially distributed with average 220 bytes
F	$\{33, 150\}$ ms

their physical-layer data rate to $r = 54$ Mbps for Wi-Fi and $r = 3$ Mbps for Bluetooth. Indeed, in order to achieve, e.g., a 20 Mbps-throughput in Wi-Fi a data rate of 54 Mbps should be adopted. Note that the Wi-Fi chipsets support also Wi-Fi Direct since the two technologies share the same physical layer.

5.3 Task graph generation

To describe the game graph $\mathcal{G}_T$, we take two approaches. First, we build synthetic task graphs, which emulate mobile games and allow us to easily vary the system parameters. Then we profile two real-world games and create the corresponding task graphs, so as to further verify the algorithms performance in real-world scenarios.

5.3.1 Synthetic task graphs

We generate random task graphs with a given number of tasks $|\mathcal{T}|$ and a given average out-degree of each task $d_{\mathcal{T}}$. The graph generation process starts with a graph of $|\mathcal{T}|$ isolated nodes. Then an ordered pair of tasks (t_1, t_2) is randomly selected and a directed edge from t_1 to t_2 is added to the graph. Such select-and-add operation continues until the number of distinct edges in the graph equals $|\mathcal{T}| \cdot d_{\mathcal{T}}$, so as to achieve the expected average degree. The amount of data exchanged between each pair of tasks is exponentially distributed with 500 bytes as the mean value. The frame period is fixed to either 33 ms (for a frame rate equal to 30 fps), or 150 ms (for a generic real-time application). All the settings for this scenario are reported in Table 7.

5.3.2 Real task graphs

We select two open-source `python` games: MICshooter [22] and Minecraft clone [23]. MICshooter is a classic single-person arcade space shooting game, while Minecraft is a clone of the popular multi-player sandbox game. For both games, we first analyze the source code and partition the game into relevant tasks at the class level. In our experiment, we partition MICshooter into 6 tasks and Minecraft clone into 5 tasks. Then we run the game and get the real task graphs for both games as follows. First, the game call graph can be obtained in real-time using the `pycallgraph` module [24]. We then process the call graph by categorizing the method calls at class level to get the actual task graph

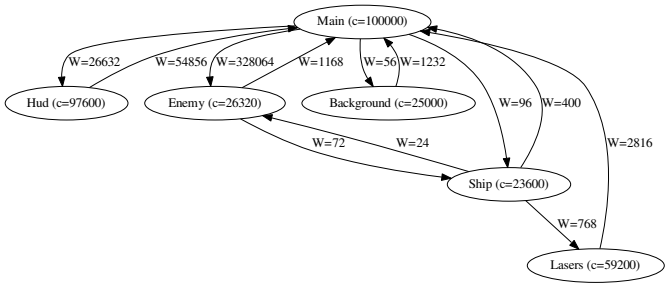

Fig. 4: Experimental task graph $\mathcal{G}_T$ for MICshooter: c is expressed in CPU cycles and W in bytes.

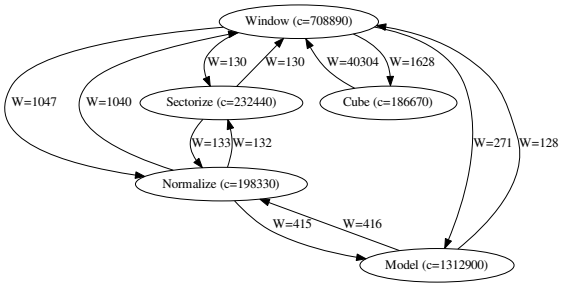

Fig. 5: Experimental task graph $\mathcal{G}_T$ for Minecraft clone: c is expressed in CPU cycles and W in bytes.

TABLE 8: Average approximation ratio for Wi-Fi Direct (top) and Bluetooth (bottom) connections, $|\mathcal{N}| \in \{4, 6, 8, 10, 12\}$ and $|\mathcal{O}| = 0$

Alg. \ $\lvert\mathcal{T}\rvert$	4	6	8	10
TAME	1.01-1.01	1.01-1.02	1.01-1.02	1.02-1.03
BESTFIT	3.16-3.40	4.59-4.77	5.81-6.04	6.80-6.92
MERGEGRAPH	1.23-1.30	1.45-1.48	1.46-1.51	1.48-1.56
Alg. \ $\lvert\mathcal{T}\rvert$	4	6	8	10
TAME	1.01-1.01	1.01-1.01	1.02-1.04	1.09-1.12
BESTFIT	3.21-3.46	4.52-4.80	5.78-5.93	6.53-6.61
MERGEGRAPH	1.21-1.32	1.54-1.59	2.25-2.33	4.75-5.01

$\mathcal{G}_T$. The average CPU requirement c_t of each task t in a frame period can be approximated by the total CPU time provided by `cProfile` [25] divided by the number of frames. The average amount of exchanged data between tasks is obtained by parsing the source code and setting up interception points to measure the size of exchanged data, considering the input variables and the output ones of each call. Notably, to obtain the actual size of the whole data structure referred by a variable, we adopt the python module `asizeof` [26], which recursively measures the referents of a data structure.

Fig. 4 and 5 depict the resulting task graphs, highlighting the computation requirement (c) and the communication requirement (W) of each task.

6 NUMERICAL RESULTS

Here we compare the performance of the TAME algorithm against our benchmark schemes, under both synthetic and real-world scenarios.

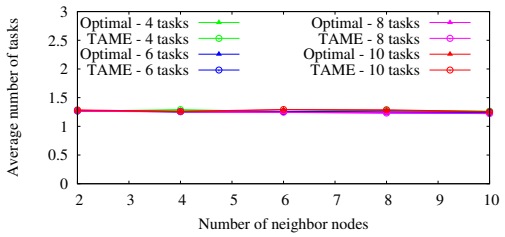

Fig. 6: Tasks running locally on player's node, for Wi-Fi Direct communications.

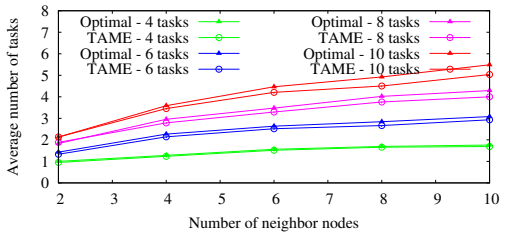

Fig. 7: Tasks offloaded to neighbor nodes, for Wi-Fi Direct communications.

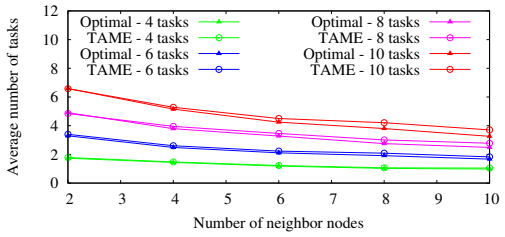

Fig. 8: Tasks offloaded to the game server, for Wi-Fi Direct communications.

6.1 Synthetic scenarios

We first use synthetic scenarios since they allow us to easily vary all the experimental settings. In particular, we first investigate the effect of the number of nodes and tasks, under Wi-Fi communications in Sec. 6.1.1 and under Bluetooth communications in Sec. 6.1.2. Then, in Sec. 6.1.3, we study the impact of different communication delays between the player's device and the game server; this allows us to better understand the benefit of task offloading in the case of different architectures such as cloud, edge and fog computing. Finally, in Sec. 6.1.4 we study how the system performance varies as the number of objects shared among the nodes changes.

6.1.1 Scenario with Wi-Fi Direct communications

Initially, we neglect the process of object retrieval by setting $|\mathcal{O}| = 0$ and assume that the game server is "close" to the PoA, i.e., we set $D_{ac} = 1$ ms. We fix $F = 33$ ms, corresponding to the common 30 fps refresh rate. Table 8(top) reports the average approximation ratio of TAME, BESTFIT and GRAPHMERGE, assuming only Wi-Fi Direct D2D communications. Given a number of tasks $|\mathcal{T}|$ and a number of nodes $|\mathcal{N}|$, we run 1000 different experiments and evaluate the approximation ratio averaged over 1000 instances. In

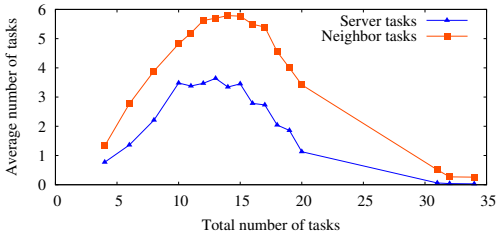

Fig. 9: Offloaded tasks for Wi-Fi Direct communications and $F = 33$ ms

Table 8, we fix $|\mathcal{T}|$ and vary $|\mathcal{N}| \in \{4, 6, 8, 10, 12\}$; in each cell, for each $|\mathcal{T}|$, we report the minimum and maximum (average) approximation ratio obtained by varying $|\mathcal{N}|$.

Observe that the TAME algorithm is always very close to the optimal solution, with a maximum approximation ratio of 1.03, and always outperforms the other algorithms. On the contrary, BESTFIT gives the worst performance, since it concentrates the tasks in one (n_0) or few nodes thus increasing their energy consumption, while the other algorithms tend to balance the task load across multiple nodes. As expected, this problem is exacerbated as $|\mathcal{T}|$ increases. With regard to GRAPHMERGE, it behaves worse than TAME, because it assumes that the problem is mainly dominated by communication energy costs (which is not the case in this scenario), hence it tends to co-locate different tasks on the same node, regardless their computation energy cost. TAME instead is able to adapt its choices to the dominant energy contribution.

To better understand the behavior of TAME and OPTIMAL, we show how the tasks are distributed across the local node (Fig. 6), the neighbor nodes (Fig. 7) and the game server (Fig. 8), in the same scenario as for Table 8(top). We report the results as functions of the number of neighbor nodes (which is equal to $|\mathcal{N}| - 2$). TAME behaves almost identically to OPTIMAL in terms of number of offloaded tasks toward the neighbor nodes and the game server, and this justifies the approximation ratio very close to 1 reported in Table 8(top). In particular, according to Fig. 6, the number of local tasks is 1 most of the times, and 2 in all other cases, independently from the number of neighbor nodes, thus TAME is very effective in offloading tasks. By comparing Fig. 7 to Fig. 8 as the number of neighbor mobile nodes increases, we note that fewer tasks are offloaded to the game server while more tasks are delegated to the neighbor nodes. This is because a higher number of neighbor mobile nodes provides more choices to offload tasks. Notably, the number of offloaded tasks per node is on average at most one, thus TAME tends to distribute equally the load across all neighbor nodes, although some of them do not host any task since offloading is not convenient. All other tasks are offloaded to the game server.

Figs. 9 and 10 show how the tasks are offloaded across the game server and the neighbor nodes, as the number of tasks increases. Both small ($F = 33$ ms) and large ($F = 150$ ms) values of the frame period are considered, while the other parameters are set as reported in Table 5.3. The results refer only to TAME since for such large numbers of tasks it is not possible to run the solver and obtain

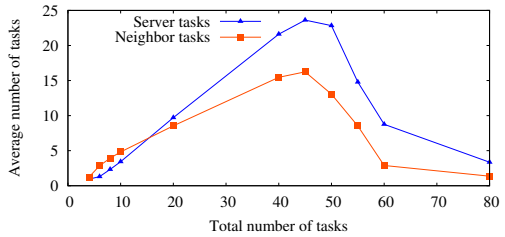

Fig. 10: Offloaded tasks for Wi-Fi Direct communications and $F = 150$ ms

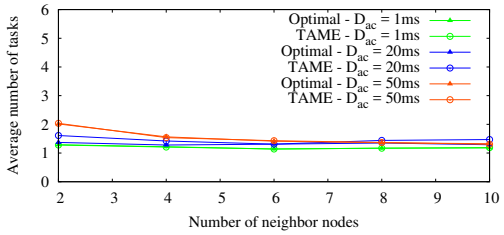

Fig. 11: Number of tasks running locally on player's node, for $|\mathcal{T}| = 6$

the optimal solution of the ILP problem. Interestingly, the number of offloaded tasks show a bell behavior: when the tasks are few, it is convenient to offload most of them, either to the game server or to the neighbor nodes. However, as the number of tasks grows further, the latency due to the data transfer between tasks allocated at different nodes becomes predominant, thus leading to a reduction in the number of offloaded tasks. When the delay requirement is very strict ($F = 33$ ms), for a number of tasks higher than 35, it is not possible to obtain any feasible solution, not even by keeping all tasks locally. Instead, relaxing the time requirement as in Fig. 10, it is still convenient to offload some tasks (even in the case of a total of 80 tasks).

In general, the level of granularity in the task partition affects the efficiency of the offloading scheme and it is clear that it is better to partition into aggregation of tasks that limit the communication needs between the different partitions. The optimal way to partition a given application is outside the scope of this work.

6.1.2 Scenario with Bluetooth communications

Table 8(bottom) shows the average approximation ratio when mobile nodes communicate via Bluetooth, and Wi-Fi is used to communicate with the PoA. In this case too, TAME greatly outperforms the other algorithms. Note that TAME performs a little worse for 10 tasks, compared to the Wi-Fi Direct case in Table 8(top). Indeed, Wi-Fi Direct and Bluetooth have quite different link speeds. Since in our experiment the link speed of Wi-Fi Direct (20 Mbps) is 20 times higher than that of Bluetooth (1 Mbps), the response delay constraint becomes more critical in the latter case. As a result, TAME is reluctant to offload tasks to the neighbor nodes and inclined to keep more tasks locally on the player's node n_0. This behavior is exacerbated by a larger number of tasks, since the bandwidth required for the communication between tasks increases. Similarly, GRAPHMERGE performs worse when adopting Bluetooth than in the case of Wi-Fi Direct communications.

6.1.3 Varying the delay between the PoA and the game server

We now investigate the impact of the propagation delay D_{ac} between the PoA and the game server. We consider $|\mathcal{T}| = 6$ tasks, and the response delay constraint $F = 150$ ms, which is the maximum lag for real-time applications. We then assume Wi-Fi Direct connections between mobile nodes and vary $D_{ac} \in \{1, 20, 50\}$ ms. Figs. 11-13 show how the tasks are offloaded across the player's node, the neighbor

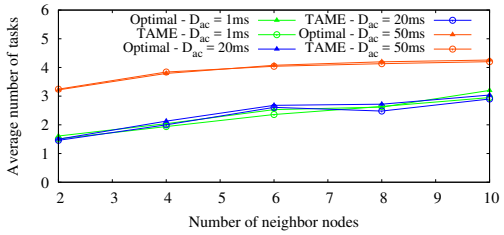

Fig. 12: Number of tasks offloaded to the neighbor nodes, for $|\mathcal{T}| = 6$

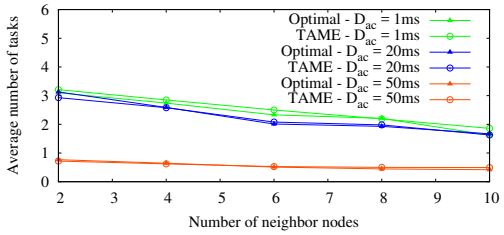

Fig. 13: Number of tasks offloaded to the game server, for $|\mathcal{T}| = 6$

nodes and the game server. As in the previous scenario, TAME behaves almost identically to OPTIMAL. In particular, the number of tasks running on the player's node is always very low as the policy tends to distribute the tasks across all mobile nodes. With regard to the average number of tasks offloaded to the game server (see Fig. 13), it is almost the same for D_{ac} equal to 1 ms and 20 ms, while it significantly decreases for $D_{ac} = 50$ ms. Indeed, in the latter case no tasks, or at most one task, can be offloaded to the game server to meet the strict response delay, independently from the number of neighbor nodes.

6.1.4 Multiple objects

To investigate the effect of object retrieval, we now set the number of objects $|\mathcal{O}| = 10$ and their size to $s_0 = 1000$ bytes. During every run of simulation, each object is located in one mobile node selected at random. Each task requires a given object with probability 0.5, thus it requires on average 5 objects. Then we run exactly the same experiments performed to obtain the results in Table 8. Table 9 shows that TAME approximates within a factor 1.32 the solution obtained by OPTIMAL and still outperforms the two benchmark algorithms.

TABLE 9: Average approximation ratio for Wi-Fi Direct (top) and Bluetooth (bottom) connections, $|\mathcal{N}| \in \{4, 6, 8, 10, 12\}$ and $|\mathcal{O}| = 10$ with $s_o = 1000$ bytes

Alg. \ $\lvert\mathcal{T}\rvert$	4	6	8	10
TAME	1.04-1.06	1.10-1.16	1.13-1.19	1.16-1.27
BESTFIT	3.09-3.30	4.57-5.17	5.41-5.93	6.19-6.56
GRAPHMERGE	1.35-1.43	1.47-1.52	1.50-1.65	1.63-1.81

Alg. \ $\lvert\mathcal{T}\rvert$	4	6	8	10
TAME	1.05-1.07	1.09-1.17	1.10-1.28	1.11-1.32
BESTFIT	3.13-3.59	4.71-5.17	5.21-6.09	6.15-6.77
GRAPHMERGE	1.30-1.36	1.54-1.73	2.30-2.60	4.75-5.55

TABLE 10: Average approximation ratio for Wi-Fi Direct (top) and Bluetooth (bottom) connections and $|\mathcal{N}| \in \{4, 6, 8, 10, 12\}$

Alg. \ Games	MICshooter	Minecraft clone
TAME	1.00-1.05	1.02-1.06
BESTFIT	1.45-1.45	2.60-2.62
GRAPHMERGE	1.34-1.35	1.15-1.22

Alg. \ Games	MICshooter	Minecraft clone
TAME	1.01-1.03	1.02-1.03
BESTFIT	1.45-1.45	2.51-2.55
GRAPHMERGE	1.45-1.45	2.48-2.52

6.2 Real-world scenarios

We now consider task graphs obtained from real-world games. Table 10 shows the approximation ratio for MICshooter and Minecraft clone, considering Wi-Fi Direct and Bluetooth communications. TAME closely matches the performance of OPTIMAL, with a maximum approximation ratio equal to 1.06, and significantly outperforms BESTFIT and GRAPHMERGE. As in the previous scenarios, BESTFIT gives the worst performance, even if with a better approximation ratio than in the case of synthetic task graphs.

In conclusion, TAME approximates very well the optimal solution, and always outperforms BESTFIT and MERGEGRAPH in all the scenarios we tested, both synthetic and real-world task graphs.

7 RELATED WORK

Offloading for mobile computing. The problem of mobile computation offloading has been widely studied. [27] exploits task offloading to achieve high throughput for data stream applications. [28] proposes a task offloading and scheduling framework to minimize energy consumption. [12] proves that the problem of multi-user computation offloading converges to a Nash Equilibrium, and proposes a distributed approach based on game theory to decide local or cloud execution for the tasks of each user. [17] proposes a mobile task offloading algorithm that merges computationally light but heavily communication tasks into super-tasks, and offloads them to the most suitable mobile nodes. However, all the above studies only address the mobile edge/cloud offloading problem without exploiting the available resource in the mobile fog. One of the algorithms used for our performance comparison, named MERGEGRAPH and presented in Sec. 5.1, is an adaptation of the algorithm proposed in [17] to our hybrid edge/fog scenario.

Other works aim at implementing task offloading platforms in practice, which can be considered as possible *key enabling technologies to support our proposed offloading scheme in IMG scenarios*. Offloading software platforms are available both for both Windows smartphones [29] and for Android smartphones [10], [11], [29], [30], [31], [32]. Interestingly, [11], [32] are able to support offloading without affecting the source code, thus seamlessly for the developer.

Related to the integration of cloud and edge/fog computing to offload tasks for mobile applications, [33] implements a framework to offload a face recognition application. Notably, face recognition applications use predefined task graphs, which, compared to those of mobile games, are not real-time. Besides, the energy cost is not considered since face recognition algorithms are computation intensive and are always offloaded. [34] proposes a combined cloud and fog/edge architecture, where the cloud servers, the neighbor mobile nodes and the edge servers are all considered as candidates for task offloading. An ILP optimization problem to allocate tasks is formulated, with the objective of minimizing latency. The application is modeled as a set of independent tasks, thus, unlike our work, it does not consider the dependencies and the communication between different tasks. Furthermore, [34] considers neither energy costs nor communication latency.

Mobile gaming. [35] proposes an approach to offload part of the artificial intelligence (AI) tasks to external servers and exploit their higher computational power, so as to allow more complex AI components, hereby more interesting and challenging games to play. Similarly, [36] studies the performance of AI offloading with various network latency and evaluates the effectiveness of dead reckoning algorithms to mitigate the performance degradation due to large delays. Furthermore, [37] exploits the general-purposed GPUs to offload the AI components of chess games. Besides, [38] studies the performance of offloading the AI components to servers available in the local network by varying the complexity and number of AIs. Obviously, all these works offload tasks at the component level. In contrast, our work offloads tasks at a finer granularity (class level), thus providing higher offloading flexibility.

Instead of studying solely the possibility of offloading AI components, [39] proposes a framework to decompose a mobile game into tasks and adaptively migrate them between the player's device and the cloud server. The main difference with respect to our work is that, instead of just considering the cloud, our model exploits neighbor mobile devices. [8] and [9] exploit the resource of both the cloud and the edge servers to enhance the quality of mobile games. Similarly, [9] envisions servers available at the user premises as fog servers in charge of video rendering and transmission, so that the user coverage increases while transmission delay and bandwidth consumption decrease. Unlike [8], [9], our work exploits neighbor mobile devices like smartphones and tablets to offload tasks, thus achieving a higher level of pervasiveness thanks to the popularity of such devices.

8 CONCLUSIONS

We devised a combined Integrated Mobile Gaming (IMG) scheme that efficiently offloads some internal tasks of a

game running on the player's device toward neighbor mobile nodes or the game server. We first formalized the problem of energy-aware task allocation as an ILP problem, which minimizes the maximum energy consumption across all the mobile nodes while accounting for the communication and computation costs involved in running and migrating tasks. Then, in light of the problem complexity, we proposed TAME, an algorithm that, at each iteration, adapts its allocation decisions based on the major factor (either communication or computation) contributing to energy consumption. We evaluated TAME in the case of synthetic and real-world scenarios. Our numerical results show that TAME closely approximates the optimal solution in all considered scenarios and outperforms other state-of-art algorithms. They also highlight the advantages of task offloading toward neighbor mobile nodes, especially when communication latency with the game server is significant. Importantly, thanks to the recent availability of offloading platforms, our TAME algorithm can be integrated in real platforms for IMG, enabling highly pervasiveness and efficient real-time mobile gaming.

ACKNOWLEDGMENT

The work was supported by Fondazione CRT under the "La Ricerca dei Talenti" grant, through the MIMOSE project.

REFERENCES

[1] O. Soliman, A. Rezgui, H. Soliman, and N. Manea, "Mobile cloud gaming: Issues and challenges," in *MobiWis*. Springer, 2013.
[2] Newzoo Free 2016 Global Games Market Report. [Online]. Available: https://newzoo.com/insights/articles/
[3] Research Report on Global Market. [Online]. Available: https://www.wiseguyreports.com/
[4] Onlive. inc. [Online]. Available: http://www.onlive.com/
[5] Gaikai. inc. [Online]. Available: http://www.gaikai.com/
[6] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "GamingAnywhere: an open cloud gaming system," in *ACM MMSys*, 2013.
[7] Y.-T. Lee, K.-T. Chen, H.-I. Su, and C.-L. Lei, "Are all games equally cloud-gaming-friendly?: an electromyographic approach," in *IEEE NetGames*, 2012.
[8] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "A hybrid edge-cloud architecture for reducing on-demand gaming latency," *Multimedia Systems*, vol. 20, no. 5, 2014.
[9] Y. Lin and H. Shen, "CloudFog: Leveraging fog to extend cloud gaming for thin-client MMOG with high quality of service," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, 2017.
[10] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *IEEE Infocom*, 2012.
[11] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *EuroSys*. ACM, 2011.
[12] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, 2016.
[13] L. Song, D. Niyato, Z. Han, and E. Hossain, *Wireless Device-to-Device Communications and Networks*. Cambridge University Press, 2015.
[14] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *IEEE Infocom*, 2012.
[15] N. Megiddo, "Linear programming in linear time when the dimension is fixed," *Journal of the ACM*, vol. 31, no. 1, 1984.
[16] Z. A. Mann, "Allocation of virtual machines in cloud data centers: A survey of problem models and optimization algorithms," *ACM Computing Surveys*, Aug. 2015.
[17] S. Ou, K. Yang, and J. Zhang, "An effective offloading middleware for pervasive services on mobile devices," *Pervasive and Mobile Computing*, vol. 3, no. 4, 2007.
[18] Gurobi optimization. [Online]. Available: www.gurobi.com/
[19] Verizon IP latency statistics. [Online]. Available: http://www.verizonenterprise.com/about/network/latency/
[20] Samsung Exynos 7 Octa 5433 Mobile Processor. [Online]. Available: http://www.samsung.com/semiconductor/minisite/Exynos/Solution/MobileProcessor/Exynos_7_Octa_5433.html
[21] Samsung Exynos 7 Octa 7420 Mobile Processor. [Online]. Available: http://www.samsung.com/semiconductor/minisite/Exynos/Solution/MobileProcessor/Exynos_7_Octa_7420.html
[22] MICShooter - old school space shooter. [Online]. Available: http://www.pygame.org/project-MICshooter-2342-.html
[23] Minecraft - Simple Minecraft-inspired demo written in Python and Pyglet. [Online]. Available: https://github.com/fogleman/Minecraft
[24] Python call graph. [Online]. Available: http://pycallgraph.slowchop.com/en/master/
[25] profile and cProfile Module Reference. [Online]. Available: https://docs.python.org/2/library/profile.html#module-cProfile
[26] Sizing individual objects. [Online]. Available: https://pythonhosted.org/Pympler/asizeof.html
[27] J. Ghaderi, S. Shakkottai, and R. Srikant, "Scheduling storms and streams in the cloud," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 1, no. 4, 2016.
[28] S. E. Mahmoodi, R. Uma, and K. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Transactions on Cloud Computing*, 2016.
[29] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in *MobiSys*. ACM, 2010.
[30] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *MobiCASE*. Springer, 2010.
[31] H. Qian and D. Andresen, "An energy-saving task scheduler for mobile devices," in *IEEE/ACIS ICIS*, 2015.
[32] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang, "Refactoring android java code for on-demand computation offloading," in *SIGPLAN Notices*, vol. 47, no. 10. ACM, 2012.
[33] F. A. Silva, P. Maciel, R. Matos *et al.*, "A scheduler for mobile cloud based on weighted metrics and dynamic context evaluation," in *ACM SAC*, 2015.
[34] V. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, and G. Tashakor, "Handling service allocation in combined fog-cloud scenarios," in *IEEE ICC*, 2016.
[35] J. R. Douceur, J. R. Lorch, F. Uyeda, and R. C. Wood, "Enhancing game-server AI with distributed client computation," *ACM NOSSDAV*, 2007.
[36] J. Bai, D. Seah, J. Yong, and B. Leong, "Offloading AI for peer-to-peer games with dead reckoning," in *IPTPS*, 2009.
[37] K. Choi, J. Lee, Y. Kim, S. Kang, and H. Han, "Feasibility of the computation task offloading to GPGPU-enabled devices in mobile cloud," in *IEEE ICCAC*, 2015.
[38] G. M. Yeung. (2015) Cloud gaming and simulation in distributed systems. [Online]. Available: https://minerva.leeds.ac.uk/bbcswebdav/orgs/SCH_Computing/FYProj/reports/1415/YEUNG.pdf
[39] W. Cai, H. C. Chan, X. Wang, and V. C. Leung, "Cognitive resource optimization for the decomposed cloud gaming platform," *IEEE Trans. on Circuits and Systems for Video Technology*, Dec. 2015.

Tianzhu Zhang is a Ph.D. candidate at Politecnico di Torino. His research interests include SDN and mobile cloud/fog computing.

Carla-Fabiana Chiasserini (M'98, SM'09,F'18) is an Associate Professor with Politecnico di Torino. Her research interests include architectures, protocols, and performance analysis of mobile networks.

Paolo Giaccone (M'99, SM'16) is an Associate Professor with Politecnico di Torino. His main area of interest is the design of network control and optimization algorithms.